COBB

November 1992
Vol. 3 No. 11

INSIDE DOS™

Tips & techniques for MS-DOS & PC-DOS

The CONFIG.SYS file helps you get the most from your system

By Van Wolverton

Many Inside DOS *subscribers have asked us to explain the CONFIG.SYS file. In response to these requests, Van Wolverton has updated his September 1990 column, "An Overview of CONFIG.SYS," to include DOS 5 configuration commands.*

When you boot your computer, DOS expects to find two files in the root directory of your system disk (the disk that contains DOS): the CONFIG.SYS file and the AUTOEXEC.BAT file. Nearly every program or device you install either tells you to change these files or changes them for you. DOS can run without either of these files, but chances are you wouldn't be happy with the way your system runs without them.

The CONFIG.SYS file contains one or more special commands, called *configuration commands*, which tell DOS how to manage the memory, keyboard, and other devices attached to your system. The AUTOEXEC.BAT file contains ordinary DOS commands; the only thing that distinguishes the AUTOEXEC.BAT file from any other batch file is its name and location.

Each time you start DOS, it carries out the configuration commands in the CONFIG.SYS file, then the DOS commands in the AUTOEXEC.BAT file. In this article, we'll look at the purpose of the CONFIG.SYS file and some of the commands you might put into it.

Checking out the CONFIG.SYS file

You almost certainly have a file named CONFIG.SYS in the root directory of your hard disk, even if you didn't create it. Companies that sell computers usually install DOS and create a CONFIG.SYS file with a minimum set of commands on each machine they sell. If you use a computer at work, someone at your company who sets up new systems might have created the CONFIG.SYS file. Even if you set up your own system and didn't create a CONFIG.SYS file, some installation programs automatically create such a file (or modify the existing one) when you install a new program or device.

To see whether you have a CONFIG.SYS file, type:

```
C:\>type c:\config.sys
```

(That's right, type *type*; this is one of the more confusing instructions to give verbally when you're helping someone with his or her computer.)

If DOS responds *File not found*, there's no CONFIG.SYS file on your hard disk. But, most likely, you'll see something like this (the commands and order will probably be different):

```
dos = high,umb
device = c:\dos\himem.sys
device = c:\dos\emm386.exe noems
devicehigh = c:\dos\ansi.sys
buffers = 20
files = 40
stacks = 9,256
shell = c:\command.com c:\ /e:512 /p
```

Don't worry if these commands look unfamiliar. Unlike the more common DOS commands, such as DIR or COPY, you don't use configuration commands every day, every week, or even every month. But when you install a new program or device, or want to tune up your system, it helps to understand the purpose of configuration commands and know how to use them.

IN THIS ISSUE

Configuration commands

There are too many configuration commands to describe in a single article, but the following brief descriptions will give you a basic idea of some of the most important commands:

- BREAK controls the circumstances under which you can stop a command or program by pressing [Ctrl][Break] (or [Ctrl]C).
- BUFFERS specifies the amount of memory DOS sets aside to handle disk files. Up to a point, the more buffers you specify, the faster DOS can read and write files.
- COUNTRY tells DOS which language characteristics to follow, such as keyboard layout and time and date formats.
- DEVICE loads the specified device driver into conventional memory.
- DEVICEHIGH loads a device driver into high memory. To use a DEVICEHIGH statement, you must first use the DOS=HIGH command.
- DRIVPARM tells DOS the operating characteristics (such as number of tracks and sectors per track) of a nonstandard disk drive.
- FILES specifies the number of files DOS can use at one time. Certain programs, such as database programs, may need to use 20 or more files. (Some older programs also use a command named FCBS to specify the same thing.)
- INSTALL loads DOS commands that remain in memory, such as FASTOPEN, KEYBOARD, NLSFUNC, and SHARE.
- LASTDRIVE specifies the highest drive letter that DOS will recognize. You may need this command if you use several disk drives—including a disk drive simulated in memory using VDISK.SYS or RAMDISK.SYS—or a large hard disk partitioned as several volumes.
- SHELL tells DOS where to find the command interpreter (usually COMMAND.COM). It also lets you increase the size of the *environment*, an area of memory that DOS uses.
- STACKS lets you control how much memory DOS reserves for its own temporary use.
- SWITCHES lets you prevent DOS from using the additional keys, such as [F11] and [F12], on the enhanced keyboard. You'll only need this command if you use a program that's incompatible with the [F11] and [F12] keys.

INSIDE DOS™

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices

Domestic	$49/yr. ($6.00 each)
Outside U.S.	$69/yr. ($8.50 each)

Phone

Toll free	(800) 223-8720
Local	(502) 491-1900
Customer Relations FAX	(502) 491-8050
Editorial Department FAX	(502) 491-4200

Address

You may address tips, special requests, and other correspondence to:

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to:

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster

Second class postage is paid in Louisville, KY. Send address changes to:

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Back Issues

To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $6.00 each, $8.50 outside the U.S., and you can pay with MasterCard, VISA, Discover, or American Express, or we can bill you. Please identify the issue you want by the month and year it was published. Customer Relations can also provide you with an issue-by-issue listing of all the articles that have appeared in *Inside DOS*.

Staff

Editor-in-Chief	Suzanne Thornberry
Contributing Editors	Van Wolverton
	David Reid
Editing	Linda Watkins
	Polly Blakemore
Production	Marguerite Stith
Design	Karl Feige
Publications Manager	Elayne Noltemeyer
Managing Editor	Jim Welp
Circulation Manager	Brent Shean
Publications Director	Linda Baughman
Editorial Director	Jeff Yocom
Publisher	Douglas Cobb

If you want to learn more about any of these or other configuration commands, see the DOS manual or one of the many DOS books available. In addition to these sources, *Inside DOS* will cover more of the configuration commands in future articles.

Driver training for DOS

When you attach a new device to your system, such as a scanner or CD-ROM drive, you usually must also copy to your hard disk a special program, called a *device driver*, that tells DOS how to control (or *drive*) the device. You load this program by including a DEVICE command in the CONFIG.SYS file.

When you buy a new device, it almost always includes the device driver. Many devices also come with an installation program, which will automatically add the DEVICE command to the CONFIG.SYS file (and, if needed, one or more commands to the AUTOEXEC.BAT file). This assistance can be a mixed blessing, though, because some installation programs don't modify the CONFIG.SYS file and the AUTOEXEC.BAT file intelligently. Depending on how much thought went into the design of the installation program, adding your new device can cause other devices—or, in rare instances, your entire system—to stop working properly. A few installation programs create a new CONFIG.SYS file and save your existing one under a different name, while others add the new commands to the beginning or end of the CONFIG.SYS file without regard to what comes before or after. Luckily, this sort of unhelpful help is pretty rare. In addition to the device drivers that come with nonstandard devices, DOS itself includes several device drivers:

- ANSI.SYS activates a series of commands that you—or a program—can use to control the color of the display, position the cursor, and assign special meanings to keys. Many programs won't operate properly unless there is a DEVICE command in the CONFIG.SYS file that loads ANSI.SYS.
- COUNTRY.SYS controls how DOS deals with different languages, including such characteristics as the keyboard layout and time and date formats.
- DISPLAY.SYS controls the character set used by the display for different languages.
- DRIVER.SYS tells DOS how to use nonstandard diskette drives.
- EGA.SYS lets you use the Task Swapper in the DOS Shell with an EGA display.
- HIMEM.SYS tells DOS how to use extended memory (memory above 1 megabyte).
- KEYBOARD.SYS defines the keyboard layout for different languages.
- PRINTER.SYS defines the character set used by several IBM printers to support different languages.
- RAMDRIVE.SYS (in MS-DOS) or VDISK.SYS (in PC-DOS) define simulated fixed disk drives in memory.
- SMARTDRV.SYS sets aside some memory for use as a disk cache.

The general form of the DEVICE command is

```
device=filename
```

where *filename* is the path and filename of the device driver program. (Some device drivers, such as ANSI.SYS, may allow or even require additional parameters of their own.)

A typical CONFIG.SYS file

If there's no CONFIG.SYS file on your system disk and your system uses an 80386 or 80486 processor, you're not taking advantage of the memory-management capabilities of DOS 5. You should create a CONFIG.SYS file that includes at least the following configuration commands:

```
dos = high,umb
device = c:\dos\himem.sys
device = c:\dos\emm386.exe noems
buffers = 20
files = 20
shell = c:\command.com /e:512 /p
break on
```

(The commands shown in color are for 386 and 486 computers with extended memory. If you attempt to use these commands, please see the trouble-shooting tips on page 4.) You can use the DOS Editor to create or change the CONFIG.SYS file because it doesn't put any special formatting codes in the file. If you use your word processor to work with the CONFIG.SYS file, be sure you save the file as *text-only* or *ASCII.*

Even if you have a CONFIG.SYS file, you might want to check its contents if your system is an 80386 or 80486. Make sure that the file includes at least the first three memory-management commands shown in the preceding example. Depending on the size and speed of your fixed disk and the types of programs you use, you may want to increase the numbers of both FILES and BUFFERS shown in the preceding example.

Should memory fail you

Most configuration commands set aside memory for a particular use, reducing the amount of memory available for DOS and application programs. Terminate-and-stay-resident programs, such as DOSKEY, consume even more memory. Where 640 Kb once seemed limitless, now it's not uncommon for so much memory to be reserved that application programs are significantly slowed or even prevented from running. That's why

DOS 5's ability to use high memory is so important.

If you start running short of conventional memory despite the memory-management features of DOS 5, you don't have many choices to free memory: You can eliminate some TSRs, but you've probably become accustomed to their convenience; you can change or eliminate some configuration commands, but this can reduce the performance of your system or make a device inoperable.

Rather than making a permanent change to your system's configuration, there's a third alternative: You can create two or more alternate CONFIG.SYS files that define different configurations with different memory requirements, then write a batch file that selects the configuration that best matches the work you're planning to do and restarts your system. We'll describe this technique in a future issue of *Inside DOS*.

Tailoring the CONFIG.SYS file to your hardware and software requires more understanding of how the computer works than using a word processor or spreadsheet program. Unfortunately, this is the price you pay for adding more hardware and using new programs. Even with smarter, more well-behaved installation programs, there will always be that special circumstance you can handle only if you know how to deal with the CONFIG.SYS file.

Troubleshooting tips

Since the CONFIG.SYS file is the first file your computer reads, using a configuration command that's incompatible with your system can prevent your system from booting. For this reason, it's extremely important that you have an emergency system diskette on hand *before* you modify your CONFIG.SYS file. We described how to create the diskette in the article "Protecting Your Data and Equipment from Disaster," which appeared in the July 1992 issue. If you haven't created this diskette, you can create a basic one by placing a blank diskette in the A: drive and entering the command *format a: /s*. Then, copy your current AUTOEXEC.BAT and CONFIG.SYS files to the new boot diskette. Finally, test your boot diskette by leaving it in the A: drive and pressing [Ctrl][Alt][Delete]. The A: drive light will stay on as your system reads files from the boot disk. Then, you should see the DOS prompt. If the diskette doesn't work, try creating another and repeating the test.

In an emergency, you can boot your system from the diskette, then edit the CONFIG.SYS file on your C: drive. First, check for typos, especially in the device statements. If you see no errors, try deleting the *noems* specification from the statement that installs the EMM386.EXE driver. You also might try moving the first line, *dos=high,umb*, after the statements installing the EMM386.EXE and HIMEM.SYS drivers. Then, remove the boot diskette from the A: drive and try to reboot from the C: drive.

The following articles discuss different aspects of the CONFIG.SYS file:

"Understanding Your Computer's Memory," July 1992

"Reading the Messages Generated by Your CONFIG.SYS," June 1992

For information on ordering back issues, see the masthead on page 2.

BATCH FILE TECHNIQUE

FINDIT.BAT lets you locate files based on their content

Bill Woodruff developed the batch file presented in this article.

Unless you're lucky enough to possess a photographic memory, you've probably spent more time than you'd like searching for a particular document on your hard disk. You can't be very descriptive with an eight-character filename, even if you use the three-character extension. In the worst case, you might resort to opening suspected files in your word processor and using its "find" capabilities to search for the text.

Bill Woodruff, who subscribes to our sister publication, *The DOS Authority*, has created FINDIT.BAT, shown in Figure A. This batch file helps you find the files you need—without opening a bunch of files from within an application. When you type *findit*, followed by a string of up to three words, the batch file will look through the files in the current directory and list the ones containing the word or phrase you specified.

We realize that FINDIT.BAT might seem a bit intimidating at first glance. Indeed, there are a few tricks involved in getting the FIND command to do so much work for you. As you'll see, however, many of the lines in FINDIT.BAT simply echo instructions to the user. We think you'll find that creating the batch file is well worth the effort, since Mr. Woodruff's FINDIT.BAT will give you a utility that might save you a lot of time and effort.

As with any batch file, you can create FINDIT.BAT using the DOS 5 Editor or an ASCII-compatible word processor. Now, let's look more closely at what FINDIT.BAT does.

How FINDIT.BAT works

Although FINDIT.BAT is long, it basically does four things:

1 Checks each file in the subdirectory to see if the file contains the text string you specified

2 Lists the name of each file in the directory, followed by the number of times the text string occurs in the file (The file that contains this list is called FOUND.FYL.)

3 Searches the list to find the lines that contain the numbers 1, 2, 3, 4, 5, 6, 7, 8, or 9. This weeds out the files that contain "0" occurrences of the text string. (This step creates a new file called GOT_IT.FYL.)

4 Displays the final list of files contained in GOT_IT.FYL

As you look at the batch file more closely, you'll find that many of the commands are slight variations on others used in the file. This repetition processes the files in the entire directory and refines the final list.

The first few lines of the batch file are quite common. As you know, *@echo off* tells DOS not to display each command as the batch file issues it. On the next line, we've used the REM command at the beginning of FINDIT.BAT to briefly describe the batch file. Then, the CLS command simply clears the screen. The next line

```
md finder
```

creates a new subdirectory that will contain the temporary files FINDIT.BAT needs. (If you already have a subdirectory on your hard drive named FINDER, you should select another name for this directory. Then, be sure to use the directory name you've chosen throughout FINDIT.BAT.)

The fifth line

```
if %3"==" goto two
```

checks to see if you entered a third word when you ran the batch file. This hypothetical third word is represented by the

Figure A

```
@echo off
rem Bill Woodruff's FINDIT.BAT searches for a string of up to three words
cls
md finder
if %3"==" goto two
set stuph=phrase "%1 %2 %3"
echo Searching for "%1 %2 %3" . . . please wait
for %%a in (*.*) do find /i /c "%1 %2 %3" %%a >> finder\found.fyl
goto zoom
:two
if %2"==" goto one
set stuph=phrase "%1 %2"
echo Searching for "%1 %2" . . . please wait
for %%a in (*.*) do find /i /c "%1 %2" %%a >> finder\found.fyl
goto zoom
:one
set stuph=word "%1"
echo Searching for "%1" . . . please wait
for %%a in (*.*) do find /i /c "%1" %%a >> finder\found.fyl
:zoom
cd finder
type found.fyl | find " 1" >> got_it.fyl
type found.fyl | find " 2" >> got_it.fyl
type found.fyl | find " 3" >> got_it.fyl
type found.fyl | find " 4" >> got_it.fyl
type found.fyl | find " 5" >> got_it.fyl
type found.fyl | find " 6" >> got_it.fyl
type found.fyl | find " 7" >> got_it.fyl
type found.fyl | find " 8" >> got_it.fyl
type found.fyl | find " 9" >> got_it.fyl
cd..
cls
copy finder\got_it.fyl finder\got_it1.fyl > nul
if not exist finder\got_it1.fyl goto none
echo.
echo.
echo.
echo     The number after the filename indicates the number
echo     of times the %stuph% appeared in that file.
echo.
type finder\got_it1.fyl
echo.
echo.
echo.
goto zoom2
:none
echo.
echo.
echo.
echo     I'm sorry. The %stuph%
echo     could not be found in this subdirectory.
echo.
echo.
echo     You may wish to use the CD command to switch
echo     to another directory and repeat the search there.
echo.
echo.
echo.
:zoom2
del finder\*.fyl
rd finder
set stuph=
```

FINDIT.BAT tells you which files contain a specified string of up to three words.

parameter %3. If you haven't entered a third word, the batch file will jump to the :TWO label. However, if you've entered a third word, the file will continue with the next line:

```
set stuph=phrase "%1 %2 %3"
```

This line sets up the environment variable STUPH. An environment variable holds a value in your computer's random access memory. In this case, if you ran FINDIT.BAT by entering *findit total tapes shipped*, the batch file would set the STUPH environment variable as *phrase "total tapes shipped"*. (The word *total* replaces %1, *tapes* replaces %2, and *shipped* replaces %3.) As you'll see later, the STUPH variable will allow the batch file to display properly worded messages when it presents your search results.

The next line

```
echo Searching for "%1 %2 %3". . . please wait
```

will display a message that includes your text string. Again, returning to our example, this line would display the message *Searching for "total tapes shipped" . . . please wait*. The message will reassure you that things are progressing as FINDIT.BAT searches through large files or crowded directories.

Now the stage is set for the line that does most of the work:

```
for %%a in (*.*) do find /i /c "%1 %2 %3" %%a >>
    finder\found.fyl
```

(We've indented the second line of the FOR command because of space limitations. You would type the command on one line.) This FOR command uses the batch file variable %%a to stand for each file in the directory, which is represented by (*.*). The FOR command requires you to enclose the "set"—in this case, the wildcard file specification *.*—in parentheses. The second part of the FOR statement follows the DO command. In this case, the DO command tells DOS to carry out the following FIND command for each file in the directory:

```
find /i /c "%1 %2 %3" %%a >> finder\found.fyl
```

In this FIND command, the /I switch tells DOS to ignore uppercase and lowercase letters in the search. With this switch, you'll find occurrences of *total tapes shipped*, *Total Tapes Shipped*, *TOTAL TAPES SHIPPED*, and even *tOtal tapes shipped*. The /C switch tells the FIND command to return the filename, followed by the number of times the text represented by "%1 %2 %3"

Adding help to FINDIT.BAT

As you can see in the article "FINDIT.BAT Lets You Locate Files Based on Their Content," which begins on page 4, Bill Woodruff includes several helpful messages in FINDIT.BAT. For example, one message lets you know that the batch file is searching the directory for the search string you typed. Another message, in the :NONE section, suggests trying a new search.

Mr. Woodruff included an additional bit of help in the file he sent us. We didn't include it in FINDIT.BAT as it appears on page 5 because the article describes the basics of using the batch file. Those of you who create the file for your own use shouldn't need the extra help. However, if you share FINDIT.BAT with your colleagues, they might not understand how to use it. For example, a new user might simply type *findit* and press [Enter], expecting the batch file to prompt him or her for search terms.

You can provide more sophisticated help for novice users by adding the instructions shown in Figure A to the end of FINDIT.BAT. You'll also need to add a line near the beginning of FINDIT.BAT to trigger the help. The new line comes after the CLS command in the batch file, as shown below:

```
@echo off
rem Bill Woodruff's FINDIT.BAT searches for a string of
    up to three words
cls
if %1 "==" goto help
```

The new line, *if %1 "==" goto help*, will jump to the :HELP section when it doesn't find a first parameter. In other words, typing *findit* by itself will trigger the messages that the :HELP section generates.

Let's look a little more closely at the new section. Before you add the :HELP section, you need to append the instruction

```
goto end
```

to the last line in FINDIT.BAT, *set stuph=*. Adding *goto end* prevents the batch file from executing the :HELP section immediately after the :ZOOM2 section. If you didn't add this instruction before the :HELP section, occurs in the current file. Then, the batch file sends this information to a file named FOUND.FYL in the FINDER subdirectory. The >> symbol ensures that DOS adds the information to the end of the file without overwriting the existing information. For example, if the FIND command locates four occurrences of your search string in a file named HENLEY.DOC, it will send the following line to FOUND.FYL:

```
---------- HENLEY.DOC: 4
```

On the other hand, if the FIND command finds no occurrences of the words when it processes FREY.DOC, it will send this line to FOUND.FYL:

```
---------- FREY.DOC: 0
```

The FOR command will process each file in the current directory, replacing the batch file variable %%a with the next filename. After the FOR command processes the last file in the directory, DOS will go to the next line:

```
goto zoom
```

As you'd expect, this line tells DOS to jump to the :ZOOM label.

We'll tell you more about the :ZOOM label in a few moments. First, let's briefly look at the :TWO and :ONE labels. As their names suggest, these labels tell DOS what to do if you enter two words or one when you run the batch file.

The :TWO section, as you can see below, begins with an IF statement:

```
:two
if %2 "==" goto one
set stuph=phrase "%1 %2"
echo Searching for "%1 %2" . . . please wait
for %%a in (*.*) do find  /i /c "%1 %2" %%a >>
    finder\found.fyl
goto zoom
```

The IF statement checks to see if you entered a second parameter. If you entered only one word to search, this statement skips to the :ONE label, which we'll explain in a moment. Otherwise, the batch file continues executing the instructions in the :TWO section. The instruction after the IF statement sets the STUPH environment variable to *phrase "%1 %2"*. As in the first section of the batch file, STUPH will be used later in the section that displays messages. The next line echoes the message *Searching for*, followed by the two-word phrase you're searching for, and asks you to *please wait*. The FOR statement that

FINDIT.BAT would display help even after successful searches.

The next line is the :HELP label, which marks the beginning of the section. As you can see, the :HELP section simply consists of ECHO statements. The *echo.* lines send blank lines to the screen in order to make the display more attractive.

The other ECHO commands display the lines of your message on the screen. Of course, you can edit the message lines to customize your help system.

Finally, the :END label marks the end of the batch file. Since we included the instruction *goto end* before the :HELP section, we now must include the :END label. The two lines work together to prevent the batch file from displaying the help messages when the user enters search terms.

Figure A

```
goto end
:help
echo.
echo.
echo.
echo     The FINDIT command is designed to scan an entire subdirectory
echo     searching every file for up to three contiguous words of text.
echo.
echo.
echo     Here's the correct form of the FINDIT command:
echo.
echo            findit word word word
echo.
echo.
echo     If nothing is found, you may wish to move to another
echo     subdirectory and repeat the search there.
echo.
echo     This search is not case sensitive. It will consider
echo     Theater, theater, and THEATER to be the same word.
echo.
echo.
:end
```

You can add this help routine to the end of FINDIT.BAT to aid new users.

follows is almost identical to the one in the first section. As you can see, the only difference is that it looks for the two words represented by %1 and %2, rather than the three used in the first section. The section ends by jumping to the :ZOOM label. The :ZOOM section, which we'll look at later, further processes the report the FIND command generates.

So far, the batch file has tested to see if you entered a third parameter (*if %3 "==" goto two*) and a second parameter (*if %2"==" goto one*). If both tests are true, the batch file processes the instructions in the :ONE section instead of the first or :TWO section. As you can see, the :ONE section contains many of the same instructions as the :TWO section:

```
:one
set stuph=word "%1"
echo Searching for "%1" . . . please wait
for %%a in (*.*) do find  /i /c "%1" %%a >>
    finder\found.fyl
```

The :ONE section, however, sets the environment variable STUPH to *word "%1"*, so that the message the batch file later generates will be grammatically correct. And, of course, the :ONE section uses only the parameter %1 in the ECHO and FOR statements.

Whether you've entered a three-, two-, or one-word search string, FINDIT.BAT will eventually go to the :ZOOM label. As we said earlier, the instructions in the :ZOOM section further process the results of the FIND command. To do this, the :ZOOM section first changes to the FINDER directory with the instruction

```
cd finder
```

Then, the next nine lines refine FOUND.FYL:

```
type found.fyl | find " 1" >> got_it.fyl
type found.fyl | find " 2" >> got_it.fyl
type found.fyl | find " 3" >> got_it.fyl
type found.fyl | find " 4" >> got_it.fyl
type found.fyl | find " 5" >> got_it.fyl
type found.fyl | find " 6" >> got_it.fyl
type found.fyl | find " 7" >> got_it.fyl
type found.fyl | find " 8" >> got_it.fyl
type found.fyl | find " 9" >> got_it.fyl
```

As you can see, each line types FOUND.FYL through the FIND filter, looking for a digit from 1 to 9. When the FIND command locates a line containing the number, it sends that line to GOT_IT.FYL. In effect, this process filters out the files that contain no—or 0—occurrences of your search terms. So, while FOUND.FYL may contain the following information:

```
----------  HENLEY.DOC:  4
----------  FREY.DOC:  0
----------  SCHMIDT.DOC:  0
----------  WALSH.DOC:  1
----------  MEISNER.DOC:  0
----------  FELDER.DOC:  0
```

GOT_IT.FYL will contain only these lines:

```
----------  HENLEY.DOC:  4
----------  WALSH.DOC:  1
```

If none of the files contains the search words, GOT_IT.FYL will still be created, but it will be empty. The next lines take that possibility into account:

```
cd..
cls
copy finder\got_it.fyl finder\got_it1.fyl > nul
if not exist finder\got_it1.fyl goto none
```

First, the *cd..* command changes the batch file to the parent of the FINDER directory. Next, the batch file clears the screen. The COPY command that follows is a clever way to exclude an empty GOT_IT.FYL file, which would exist if none of the files in the directory contained your search words. If the file contains 0 bytes, DOS won't create the file GOT_IT1.FYL. On the other hand, if GOT_IT.FYL contains the name of at least one file, DOS will create GOT_IT1.FYL. Redirecting the output of this COPY command to NUL simply prevents you from seeing the message *1 file(s) copied* or *0 file(s) copied*. Next, FINDIT.BAT checks to see if GOT_IT1.FYL exists in the FINDER subdirectory. If the file does not exist, the GOTO command directs the batch file to the :NONE label. This label suggests what to do when the search words don't appear in any files in the current directory. On the other hand, if GOT_IT1.FYL exists, the batch file will continue with the next section:

```
echo.
echo.
echo.
echo  The number after the filename indicates the number
echo  of times the %stuph% appeared in that file.
echo.
type finder\got_it1.fyl
echo.
echo.
echo.
goto zoom2
```

(In this callout, we've left just one space between the ECHO commands and the message text. However, you may want to add extra space or a tab to make the messages more attractive onscreen.) After skipping three lines, the next two ECHO commands display a message explaining the information that will follow. Of course, the message replaced the environment variable represented by %stuph% with *word* or *phrase* followed by the words you're searching for. So, if you're searching for *total tapes shipped*, the batch file will display the message *The number after the filename indicates the number of times the phrase "total tapes shipped" appeared in that file*. FINDIT.BAT will skip a line, then type the GOT_IT1.FYL list to the screen. Finally, the batch file echoes three more lines to the screen before moving to the :ZOOM2 label. As we'll see in a moment, the :ZOOM2 section "cleans up" by removing the files and directories FINDIT.BAT created.

So far, we've seen what FINDIT.BAT does if it finds the files that contain your search words. But if it can't find the files, the batch file skips to the :NONE label, which is shown below:

```
:none
echo.
echo.
echo.
echo   I'm sorry. The %stuph%
echo   could not be found in this subdirectory.
echo.
echo.
echo   You may wish to use the CD command to switch
echo   to another directory and repeat the search there.
echo.
echo.
echo.
```

As you can see, the :NONE section simply echoes a few blank lines to the screen, then presents a message that your search words (contained in the STUPH environment variable) couldn't be found. Then, it suggests that you try running FINDIT.BAT again in a different directory.

After echoing its messages, the :NONE section also goes to the :ZOOM2 label:

```
:zoom2
del finder\*.fyl
rd finder
set stuph=
```

As we mentioned earlier, the :ZOOM2 section cleans up the effects of the batch file. First, the DEL command deletes all of the files in the FINDER directory (all were named with the FYL extension). Next, it removes the FINDER directory. Finally, the SET command deletes the STUPH environment variable by setting it equal to nothing.

Using FINDIT.BAT

We've already shown you some of the features built into FINDIT.BAT to make it easier to use. You get ready to search by changing to the directory you think contains the files you want to search. Then, you type *findit*, followed by up to three words.

After you've typed your search word or phrase, press [Enter]. FINDIT.BAT will present the *Searching for...* message, then pause for a few moments as it processes the files in your directory. (Processing large files or directories will take longer than processing smaller ones.) Finally, the batch file will either display the files that contain your search words, or it will present the message suggesting that you try another directory.

For example, suppose you want to search for references to turkey hash in the C:\THANKS directory. You begin your search by entering the command

```
C:\THANKS>findit turkey hash
```

FINDIT.BAT will display the message

```
Searching for "turkey hash" . . . please wait
```

After a few moments, FINDIT.BAT will present a message and a list of files containing the reference:

```
The number after the filename indicates the number of
times the phrase "turkey hash" appeared in that file.

----------  PILGRIM.DOC  1
----------  LEFTOVER.DOC  3
----------  PARADES.DOC  2
```

Notes

The FIND command also leads to another quirk in FINDIT.BAT. When you search for a single word, FINDIT.BAT will report all files containing that string of characters—even if the string is part of another word. For example, if you search for the word *here*, FINDIT.BAT will report files that contain the words *there*, *sphere*, or even *heredity*. Fortunately, you usually won't run into this problem if you search for longer words or names, such as *Hamilton* or *fluorescence*. Another way to avoid misleading reports is to search for a phrase instead of a single word.

If you use FINDIT.BAT in a directory that has filenames beginning with numbers, FINDIT.BAT may list those filenames in its report. You'll know to ignore these files, however, because the batch file will list "0" as the number of times the search string occurred in the file. Incidentally, this occurs when the batch file types FOUND.FYL through the FIND command, looking for the numerals 1 through 9.

Also, if you expect new users to try FINDIT.BAT, you might consider adding more helpful messages to the file. The accompanying sidebar beginning on page 6 shows you how to add Bill Woodruff's help routine, which appears when the user types FINDIT with no parameter.

The following articles demonstrate techniques for finding files by searching for filenames:

"Finding Missing Files with the CHKDSK and FIND Commands," May 1990

"Finding Your Files with SEARCH.BAT," September 1991

For information on ordering back issues, see the masthead on page 2.

QUICK TIP

Listing your directories when you boot up

Recently, Sam Freid of Gambrills, Maryland, noted that he begins almost every session on his PC by listing all of the directories on his hard drive. However, his current method, which he developed before he began using DOS 5, uses the FIND command to locate the names of directories from the output of the DIR command, and then sends the directory names to another file. He then has to type this file to read the names of the directories. Although his technique was an excellent way to work around the limitations of previous versions of DOS, Mr. Freid wondered if DOS 5 offered a technique that would let him view the directories in the three-column format offered by the DIR command's /W switch.

Indeed, DOS 5 offers a more direct way to view directories. With DOS 5, you don't have to use the FIND command to search for directories within the listing. Instead, you can use the /A:D switch to tell the DIR command to look for files with the directory attribute. Better still, since you don't have to send the directory names to a separate file, the /W switch will display your directories in the three-column format. So, to view all of the directories on your hard disk, you can issue the following command:

```
C:\>dir /a:d /w
```

If you find yourself issuing this command almost every time you boot up, you can append it to your AUTOEXEC.BAT file. Then, whenever you turn on your PC, you'll see the list of files on the hard drive.

Or, perhaps you find that you need the command frequently as you work with DOS. If so, you can define a new DOSKEY macro that displays the names of your directories. For example, you can create a macro named DIRS by placing the following line in your AUTOEXEC.BAT file (or your MACROS.BAT file):

```
doskey dirs=dir /a:d /w
```

With this macro installed, you can view the names of directories on your hard drive simply by typing *dirs* and pressing [Enter]. You can also use the DIRS macro from within a directory to list its subdirectories. For example, you could change to the C:\PUBLISH directory, then issue the following command:

```
C:\PUBLISH>dirs
```

to list its subdirectories.

DOS SHELL TIP

Preventing the DOS Shell from using memory it doesn't need

If you run the DOS Shell, then select the Command Prompt option from the Main section, you'll temporarily suspend the DOS Shell. However, you haven't quit the Shell, as you would if you selected the Exit command from the File menu. Instead, you can return to the Shell from the DOS prompt by typing *exit* and pressing [Enter].

Of course, it's easy to forget that the DOS Shell is only suspended, since you'll see the same DOS prompt as you normally do. When you decide to run the DOS Shell again, you might type *dosshell* and press [Enter] instead of typing *exit*.

Although running the DOSSHELL command a second time brings up the Shell again, you'll be wasting some of your system's random access memory (RAM). That's because DOS will duplicate in RAM portions of the DOSSHELL.EXE and COMMAND.COM files the second time you run DOSSHELL. In our tests, we tied up 7,984 bytes—nearly 8 Kb—by running DOSSHELL a second time.

Saving memory, of course, is the main reason to take advantage of the EXIT command when you want to return to the DOS Shell. However, using the EXIT command has another advantage: You don't have to wait as the Shell scans the files and directories on your hard disk to create the directory tree and files list.

If you have trouble remembering to use EXIT to return to the DOS Shell, you might want to try the two-fold solution that Microsoft Technical Support suggests. First, you create SHELL.BAT, shown in Figure A, which decides if you should go to the DOS Shell using the EXIT command or the DOSSHELL command. Then, you create a DOSKEY macro named DOSSHELL that runs SHELL.BAT. The beauty of the Microsoft technique is that the DOSSHELL macro takes precedence over DOSSHELL.COM—the program that normally runs the DOS Shell. Since you'll run the macro whenever you type *dosshell* and press [Enter], you won't even need to learn a different name for running the DOS Shell. Let's take a closer look at setting up this alternative way to run the DOS Shell.

Figure A

```
@echo off
if *%dosshell%*==*true* goto exit
set dosshell=true
dosshell
:exit
set dosshell=
exit
```

SHELL.BAT checks to see whether you've already run the DOS Shell.

Creating SHELL.BAT

As with any batch file, you can create SHELL.BAT using the DOS 5 Editor or any word processor that can produce ASCII-only text files. If you're using the DOS 5 Editor, change to the directory you normally use for your batch files, then type *edit shell.bat*. (In our example, we'll assume you keep your batch files in the C:\BATCH directory, which should also be on your path.) So, to create SHELL.BAT, you change to the batch directory and enter the following command:

```
C:\BATCH>edit shell.bat
```

Once you've started the Editor, simply type the seven lines shown in Figure A. Check the file for typos, then issue the Save command by selecting it from the File menu or simply by pressing [Alt]F,S. Once you've saved the file, you can issue the Exit command from the File menu ([Alt]F,X) to quit the Editor.

How SHELL.BAT works

If you're still getting acquainted with batch file language, you might wonder how SHELL.BAT works. The key to SHELL.BAT is the second line:

```
if *%dosshell%*==*true* goto exit
```

This IF statement checks to see if the DOS environment contains a variable named DOSSHELL that has been set to true. That condition is met only after you've previously run the DOS Shell. So, if the batch file finds that the DOSSHELL variable is equal to true, it jumps to the :EXIT label. (We'll look at the :EXIT section in a moment.)

On the other hand, if the condition is not met, the batch file executes the third line:

```
set dosshell=true
```

This statement sets the DOSSHELL variable to true the first time you run the DOS Shell, allowing the batch file to "remember" if you've already run the Shell.

If you have already run the DOS Shell, and the DOSSHELL variable is equal to true, the batch file carries out the instructions under the :EXIT label:

```
:exit
set dosshell=
exit
```

This SET command only deletes the DOSSHELL variable from the "child" version of COMMAND.COM you ran by selecting Command Prompt from the DOS Shell. The master COMMAND.COM, which DOS loaded when you booted up, still contains the environment variable DOSSHELL=TRUE. If you select Command Prompt later, SHELL.BAT will still "remember" to exit back to the DOS Shell. The second line issues the EXIT command to return you to the DOS Shell, which is already running. As we've explained, using the EXIT command prevents you from copying unnecessary programs into memory.

Setting up the macro

Once you've created SHELL.BAT, you're ready to assign it to a DOSKEY macro named DOSSHELL. You can place the commands for setting up the macro in your AUTOEXEC.BAT file. As usual, you'll need to use the DOS 5 Editor or an ASCII-compatible word processor to edit the AUTOEXEC.BAT file. Then, enter the command that defines the new macro:

```
doskey dosshell=c:\batch\shell.bat
```

(Of course, if you keep SHELL.BAT in a different directory, substitute its name for the C:\BATCH directory.) Now, when you reboot your PC, DOS will install the DOSSHELL macro. Whenever you type

```
dosshell
```

and press [Enter], DOS will execute the macro, running SHELL.BAT instead of the DOS Shell directly.

Note

If you read the article "Storing Your Macros and Automatically Loading Them at Bootup" from the September 1991 issue, you may have created a file called MACROS.BAT. As the article explains, you can define all of your DOSKEY macros through MACROS.BAT, which the AUTOEXEC.BAT file calls at boot up. Of course, if you've created a MACROS.BAT file, you will probably want to place the definition of the DOSSHELL macro in that file, rather than in your AUTOEXEC.BAT file.

Conclusion

In this article, we've shown you how choosing the Command Prompt option from the DOS Shell, then using the DOSSHELL command to return to the Shell can waste some of your system's RAM. You can avoid the problem by typing *exit* to return to the DOS Shell. Or, you can create a DOSKEY macro and the SHELL.BAT batch file to ensure that you use the EXIT command whenever you've previously run the DOS Shell. ■

Microsoft Technical Support
(206) 454-2030

```
DOSC:1144914 I:1406920 Q1 9211
FRANK RALLY
1796 GRACE AVE
SAN JOSE CA 95125-5620
```

Please include account number from label with any correspondence.

LETTERS

Handling filenames that contain a space

My word processor, Professional Write, created a file named MY 40. Since this filename contains a space, DOS ignores my commands when I try to copy it. How can I work with this file at the DOS prompt?

Russ Henderson
Norton, Ohio

Several DOS applications can create a filename with a space, which DOS views as "illegal." Whenever DOS sees a space, it views the next string of letters as a parameter to a command.

If you want to work with your file using regular DOS commands, you'll have to rename it in order to replace the space with a legal character. Unfortunately, you can't just issue the command *C:\DOCS>ren my 40 my_40* to replace the space with the legal underscore character. However, you can rename the file by substituting the * wildcard character for the space. So, to rename your MY 40 file, enter the command

```
C:\DOCS>ren my*40 my_40
```

While we're on the subject of filenames, let's review the characters that DOS views as "legal" in filenames. Of course, you can use any letter or number in filenames. You can also use the symbols shown in Table A.

Table A

Character	Name
!	exclamation point
@	"at" sign
#	pound or number sign
$	dollar sign
%	percent sign
^	caret
&	ampersand
(and)	parentheses
{ and }	braces
-	hyphen
_	underscore
~	tilde
'	apostrophe

Notable illegal characters include special symbols that DOS uses, such as the plus sign (+), which is used for concatenating files, and the asterisk wildcard (*). You also can't use most punctuation marks, including the comma, colon, and semicolon.

Measuring modem speed

I would like to point out an error in the August 1992 issue of *Inside DOS*. On page 3, in "Online Glossary," baud is defined as the number of bits per second that a modem can transmit. However, this definition describes BPS, another common measure of modem speed, which stands for "bits per second."

Baud rate, on the other hand, is the rate at which *symbols*—not bits—are transmitted. A single symbol typically requires several bits of information. For example, at a rate of 2400 BPS, four data bits make up one symbol. (These figures assume that the modem uses the V.22bis encoding standard, a fairly common protocol.) In this case, the data rate is actually 600 baud. You can verify the data rate by multiplying 600 baud (symbols per second) by 4 (bits per symbol), which equals 2400.

The confusion about BPS and baud probably goes back to the first widely used modems. Because of their transmitting protocol, these 300 BPS modems also transmitted symbols at a rate of 300 baud. Even today, the BPS acronym is a bit fuzzy. Using certain compression protocols, the bit rate on the phone line may be substantially lower than the bit rate seen by the user. In other cases, error control protocols may have to slow down the rate the user sees in order to work around noise on the telephone line.

Michael B. Andrews
Beaverton, Oregon

Thanks for your thorough explanation of the difference between baud and BPS. We'd also like to note that BPS is the more common measure of modem speed. So, if you're shopping for a modem, BPS usually is the speed that the advertisement or salesperson is referring to.

Printed on Halopaque: 100% acid free, made from 50% de-inked, recycled fiber including 10% post-consumer waste. This journal is 100% recyclable.